



**ДИПЛОМНАЯ РАБОТА**

**Сравнение различных библиотек  
для визуализации данных:  
Matplotlib, Seaborn и Plotly**

**Автор: Накельский Н.С.**

## Оглавление

|   |    |
|---|----|
| 1. Обзор проекта.....                               | 3  |
| 2. Структура проекта .....                          | 5  |
| 3. Обзор библиотек и реализация визуализации: ..... | 9  |
| 3.1. Matplotlib .....                               | 9  |
| 3.2. Seaborn .....                                  | 12 |
| 3.3. Plotly .....                                   | 15 |
| 4. Сравнительный анализ библиотек .....             | 18 |
| 5. Заключение .....                                 | 23 |
| 6. Приложение 1. Список необходимых библиотек.....  | 25 |

## 1. Обзор проекта

Предлагается создать набор визуализаций с использованием библиотек Matplotlib, Seaborn и Plotly, сравнить их функциональность и удобство использования.

Matplotlib, Seaborn и Plotly являются библиотеками для построения графиков массивов в Python, следовательно для реализации практической части содержания работы потребуется некоторый массив данных.

Используем данные ООО «Курильский Универсальный Комплекс» (место работы автора) по выдаче матросу средств индивидуальной защиты в период с 2014 по 2024 годы.

|    | A                                      | B                   | C    | D    | E    | F    | G    | H    | I    | J    | K    | L    | M    |
|----|--|---------------------|------|------|------|------|------|------|------|------|------|------|------|
|    | Тип СИЗ                                | Норма выдачи на год | 2014 | 2015 | 2016 | 2017 | 2018 | 2019 | 2020 | 2021 | 2022 | 2023 | 2024 |
| 1  |  |                     |      |      |      |      |      |      |      |      |      |      |      |
| 2  | Перчатки хлопчатобумажные              | 24                  | 63   | 96   | 111  | 98   | 102  | 110  | 107  | 103  | 99   | 95   | 12   |
| 3  | Перчатки прорезиненные                 | 24                  | 49   | 64   | 68   | 63   | 65   | 67   | 64   | 62   | 59   | 60   | 2    |
| 4  | Перчатки зимние                        | 2                   | 4    | 6    | 7    | 7    | 8    | 7    | 7    | 6    | 6    | 6    | 1    |
| 5  | Ботинки с усиленным носком             | 1                   | 1    | 1    | 2    | 1    | 2    | 1    | 2    | 1    | 2    | 1    | 1    |
| 6  | Сапоги рыбацкие                        | 4                   | 8    | 12   | 13   | 12   | 12   | 13   | 12   | 12   | 8    | 7    | 1    |
| 7  | Чулки в сапоги утепленные              | 2                   | 5    | 6    | 7    | 6    | 6    | 7    | 6    | 6    | 4    | 4    | 1    |
| 8  | Костюм хлопчатобумажный                | 1                   | 2    | 3    | 4    | 4    | 4    | 4    | 4    | 4    | 3    | 3    | 1    |
| 9  | Костюм рыбацкий прорезиненный          | 1                   | 4    | 6    | 8    | 7    | 8    | 8    | 8    | 7    | 4    | 4    | 1    |
| 10 | Костюм зимний утепленный влагозащитный | 1                   | 1    | 1    | 2    | 1    | 1    | 2    | 2    | 1    | 1    | 1    | 1    |
| 11 | Каскетка                               | 1                   | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    |
| 12 | Каска защитная                         | 1                   | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    |
| 13 | Шапка зимняя влагозащитная             | 1                   | 1    | 1    | 2    | 2    | 2    | 2    | 2    | 2    | 1    | 1    | 1    |
| 14 | Подшлемник утепленный                  | 1                   | 1    | 2    | 2    | 2    | 2    | 2    | 2    | 2    | 2    | 1    | 1    |

Данные представлены в виде таблицы Excel, в связи с чем потребуется использование дополнительных библиотек: `openpyxl` — для считывания данных из таблицы и `pandas` для оперирования данными и их преобразования.

Вместе с тем, для удобства сравнения библиотек и представления результатов предлагаем обернуть программу в пользовательский интерфейс с помощью библиотеки `PySide6`.

Для измерения скорости быстроедействия воспользуемся стандартной библиотекой `datetime`.

Критерии оценки исследуемых библиотек будет включать следующие пункты:

1. Особенности преобразования данных
2. Простоту построения графика без параметров стилизации
3. Полноту отображения информации
4. Быстродействие

В целях обеспечения объективности оценки библиотек остановимся на следующей методике:

1. с помощью каждой библиотеки будут построены графики двух типов:

- a. упрощенный – включает один параметр;
  - b. усложненный – включает все параметры;
- 2. быстродействие будет учитывать время между получением входных данных и завершением вывода графика;
- 3. время предобработки данных не учитывается, поскольку форма поступающих данных зависит исключительно от программы и идеи реализуемого проекта, что не влияет на реализацию функций исследуемых библиотек;
- 4. способ построения графиков будет субъективно простым;
- 5. оценки субъективного характера (красота графиков, цветовые гаммы и т.п.) отводятся на второй план и будут рассмотрены в заключении.

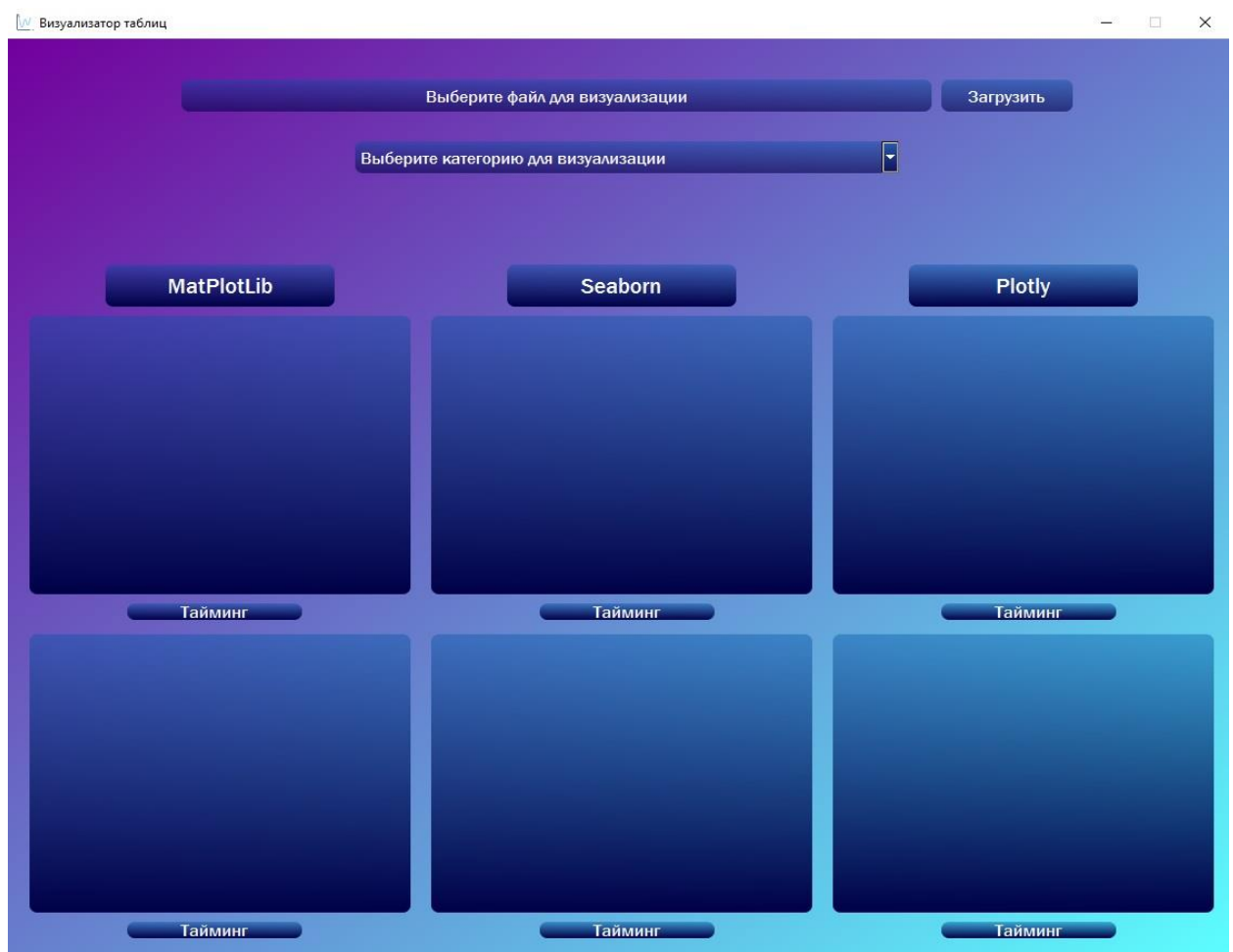
## 2. Структура проекта

Проект включает:

1. Файл `main.py`, содержащий в себе код, объединяющий GUI и backend, считывающий таблицу excel, осуществляющий предобработку данных.
2. Файл `ui_main.py`, содержащий в себе код GUI.
3. Файлы `by_matplotlib.py`, `by_plotly.py` и `by_seaborn.py`, содержащие в себе код построения и сохранения графиков, и размещенные в директории Backend.
4. Генерируемые программой файлы изображений графиков для их дальнейшего представления, размещенные в директориях `simple_plots` для простых графиков и `complicated_plots` для усложненных.

Для начала установим необходимые библиотеки с помощью `pip install` (см. Приложение 1. Список необходимых библиотек).

Далее создадим пользовательский интерфейс с помощью библиотеки PySide6 и встроенного редактора QtDesigner.



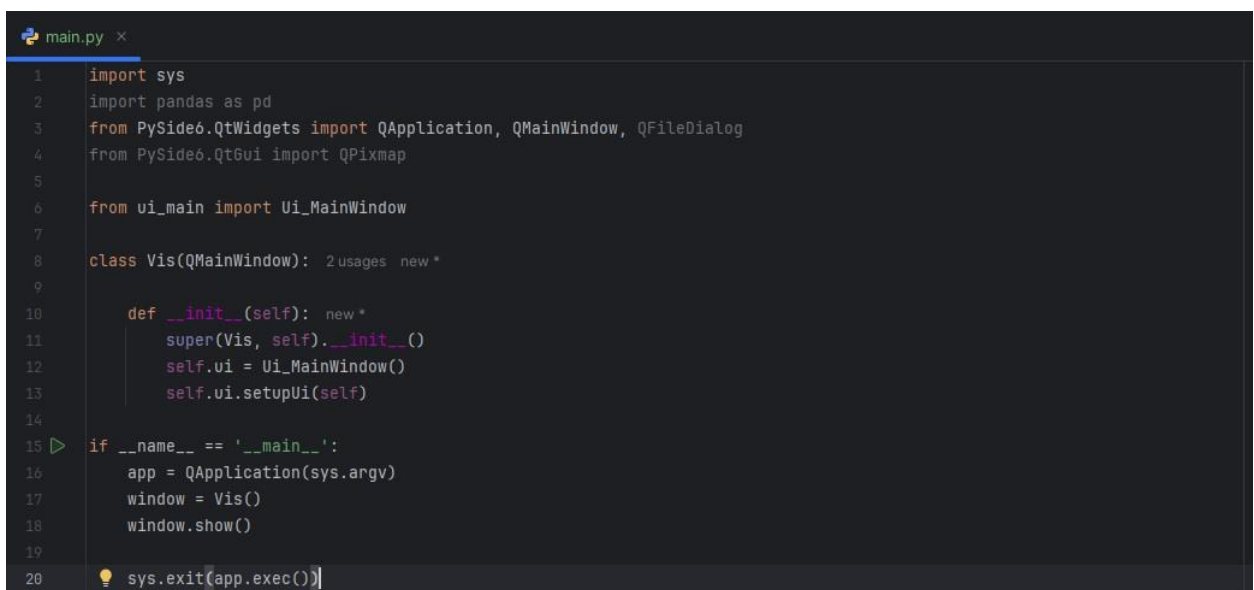


Задача интерфейса предоставить пользователю форму для выбора таблицы excel, параметра таблицы, который необходимо визуализировать, и представление графиков в виде изображений.

Также, в GUI включим сведения о быстродействии исполнения подпрограмм построения графиков.

После построения GUI и формирования файла ui\_main.ui преобразуем его в ui\_main.py для последующей работы.

Создадим файл main.py, в который импортируем необходимые для дальнейшей работы библиотеки, а также напомним класс, наследуемый от Ui\_MainWindow.



```
1 import sys
2 import pandas as pd
3 from PyQt5.QtWidgets import QApplication, QMainWindow, QFileDialog
4 from PyQt5.QtGui import QPixmap
5
6 from ui_main import Ui_MainWindow
7
8 class Vis(QMainWindow):
9     def __init__(self):
10         super(Vis, self).__init__()
11         self.ui = Ui_MainWindow()
12         self.ui.setupUi(self)
13
14 if __name__ == '__main__':
15     app = QApplication(sys.argv)
16     window = Vis()
17     window.show()
18
19 sys.exit(app.exec())
```

Также включим обработчик события, который позволяет выбрать таблицу для визуализации, объявим глобальные переменные и приступим к считыванию и предобработке данных.

Для считывания воспользуемся библиотекой pandas, при этом учитываем, что pandas самостоятельно не имеет таких возможностей, поэтому неявно используется openpyxl.

Для дальнейшей обработки данных создаем dataframe.

Из первой строки задаем список периодизации и определяем имя название для нормы выдачи.

Для реализации механизма выбора параметра, по которому будет составляться график, создаем список имен параметров и включаем его с помощью генератора списков в выпадающее окно GUI, где при выборе параметра запускается функция создания графика и вывода его на экран.

Обернем код в try-исхепт для исключения ошибок при выборе таблицы. Этого способа достаточно в рамках исследования, поскольку проверка таблицы соответствия форме не является целью работы.

```
main.py x
7
8 class Vis(QMainWindow): 2 usages new *
9
10 def __init__(self): new *
11     super(Vis, self).__init__()
12     self.ui = Ui_MainWindow()
13     self.ui.setupUi(self)
14     self.ui.browse.clicked.connect(self.browse_)
15
16 def browse_(self): 1 usage new *
17     # Объявление глобальных переменных и диалоговое окно выбора таблицы
18     global period, norm_name, names, df_mp, stat_h, df_sns_h
19     file = QFileDialog.getOpenFileName(self, caption='Открыть таблицу', filter=('Таблицы (*.xlsx)'))
20
21     # Предобработка данных
22     try:
23         table = pd.read_excel(file[0])
24         self.ui.browse_label.setText(file[0])
25         df_mp = pd.DataFrame(table)
26
27         # Задаем список периодизации и имя нормирования
28         period = []
29         for i in df_mp.columns.values[1:]:
30             if type(i) is str:
31                 norm_name = i
32             else:
33                 period.append(str(i))
34
35         # Задаем список названий строк для представления и вызываем функцию изменения вып. списка
36         names = df_mp[df_mp.columns[0]].tolist()
37         self.ui.select_type.addItem([f'{names.index(i) + 1}. {i}' for i in names])
38         self.ui.select_type.currentIndexChanged.connect(self.index_changed)
39     except:
40         print('Неверные импортируемые данные, проверьте таблицу на соответствие требованиям к форме')
```

Затем создаем функцию, срабатывающую по выбору параметра для визуализации, в которой зададим имя параметра для визуализации, получим список значений для представления по имени и норму для параметра с использованием генераторов списков.

С учетом того, что Seaborn создавался в т.ч. для быстрой и удобной работы с datafram'ами, создаем удобочитаемый им dataframe включающий массивы данных для выбранного параметра.

Для создания упрощенного графика с помощью библиотеки Plotly достаточно уже имеющихся обработанных данных.

Подготовим данные для построения усложнённых таблиц, где с помощью генератора списков определим те объемы данных, которые необходимо визуализировать.

Для Seaborn снова построим dataframe, который преобразуем для корректного отображения данных.

Для Plotly дальнейшей обработки данных не требуется.

```
def index_changed(self, i): 1 usage new *
    # Задаем имя параметра для представления
    name = names[i]

    # Получаем список значений для представления по имени и норму
    pre_list = [int(i) for i in df_mp.iloc[names.index(name)] if type(i) is not str]
    stat = pre_list[1:]
    norm_nums = [pre_list[0] for i in stat]

    # Преобразуем данные в удобочитаемый датафрейм для Seaborn
    zip_l = list(zip(norm_nums, stat, period))
    data_l = [list(i) for i in zip_l]
    df = pd.DataFrame(data_l, columns=[norm_name, 'Факт', 'Год'])

    # Подготавливаем данные для усложненных таблиц
    #Matplotlib
    stat_h=[]
    if len(stat_h) > 0:
        stat_h.clear()
    for i in names:
        pre_stat_h = [int(i) for i in df_mp.iloc[names.index(i)] if type(i) is not str]
        stat_h.append(pre_stat_h[1:])
    #Seaborn
    df_sns_h = pd.DataFrame(stat_h)
    df_sns_h.columns = [i for i in period]
    df_sns_h = df_sns_h.transpose()
    df_sns_h.columns = [i for i in names]
```

Далее приступим к обзору библиотек и созданию функций, реализующих их возможности.



### 3. Обзор библиотек и реализация визуализации:

#### 3.1. Matplotlib

Matplotlib - популярная Python-библиотека для визуализации данных. Она используется для создания любых видов графиков: линейных, круговых диаграмм, построчных гистограмм и других - в зависимости от задач.

Matplotlib была написана американским нейробиологом Джоном Хантером (англ. John Hunter) и распространяется на условиях BSD-подобной лицензии.

Изначально Хантер разработал Matplotlib во время постдокторантуры по нейробиологии для визуализации данных электрокортикографии (ЭКоГ) у пациентов с эпилепсией.

Библиотека была создана для имитации графических команд MATLAB, она не зависит от MATLAB и может использоваться в объектно-ориентированном стиле Python. Хотя Matplotlib написана на чистом Python, она активно использует NumPy и другие расширения для обеспечения высокой производительности даже для больших массивов.

Matplotlib использовалась для визуализации данных во время посадки космического аппарата «Феникс» на Марс в 2008 году и для создания первого изображения чёрной дыры.

Библиотека Matplotlib позволяет работать с данными на нескольких уровнях:

- с помощью модуля Pyplot, который рассматривает график как единое целое;

- через объектно-ориентированный интерфейс, когда каждая фигура или её часть является отдельным объектом, это позволяет выборочно менять их свойства и отображение.

Дополнительные библиотеки позволяют расширить возможности анализа данных. Например, модуль Cartopy добавляет возможность работать с картографической информацией.

Сама Matplotlib является основой для других библиотек, например Seaborn позволяет проще создавать графики и имеет больше возможностей для косметического улучшения их внешнего вида. Однако, Matplotlib - это базовая библиотека для визуализации данных, незаменимая в анализе данных.

Пакет поддерживает многие виды графиков и диаграмм:

1. Графики (англ. line plot)
2. Диаграммы рассеяния (англ. scatter plot)

3. Столбчатые диаграммы (англ. bar chart) и гистограммы (англ. histogram)
4. Круговые диаграммы (англ. pie chart)
5. Диаграммы стебель-листья (англ. stem plot)
6. Контурные графики (англ. contour plot)
7. Поля градиентов (англ. quiver)
8. Спектральные диаграммы (англ. spectrogram)

С помощью Matplotlib можно делать и анимированные изображения.

Набор поддерживаемых форматов изображений, векторных и растровых, можно получить из словаря `FigureCanvasBase.filetypes`. Типичные поддерживаемые форматы:

1. Encapsulated PostScript (EPS)
2. Enhanced Metafile (EMF)
3. JPEG
4. PDF
5. PNG
6. Postscript
7. RGBA («сырой» формат)
8. SVG
9. SVGZ
10. TIFF

Несложные трёхмерные графики можно строить с помощью набора инструментов (toolkit) `mplot3d`. Есть и другие наборы инструментов: для картографии, для работы с Excel, утилиты для GTK и другие.

Официальная страница документации библиотеки - <https://matplotlib.org>.

Приступим к созданию простейшего графика с помощью Matplotlib, для чего импортируем `os.path` для работы с файлом графического представления графика, `matplotlib.pyplot` для построения графика и `datetime` для определения быстродействия.

Создадим функцию, принимающую в себя аргументы, определяющие период, норму и статистику выбранного параметра.

Установим начало отсчета времени исполнения функции.

Зададим условие, согласно которому будет удаляться файл графического представления графика, если он уже имеется, обеспечив тем самым обновление изображения.

Создадим первую линию графика, где  $X$  – период, а  $Y$  – статистика выбранного параметра, и вторую линию, где  $Y$  – норма параметра.

Далее реализуем метод сохранения графического представления графика в выбранную директорию, после чего установим окончание отсчета времени исполнения функции и с помощью оператора return вернем значение функции в виде времени её исполнения.

```
by_matplotlib.py ×
1 import os.path
2 import matplotlib.pyplot as plt
3 import datetime
4
5 # period - период для представления
6 # norm_name - название нормы
7 # norm_nums - список значений нормы по количеству периодов
8 # name - название объекта представления
9 # stat - статистика объекта представления
10
11
12 # Строим простейший график
13 def simple_mpl(period, norm_nums, stat): 2 usages  ⚡ Hazard-sakh *
14     t_mpl1 = datetime.datetime.now()
15     if os.path.exists('simple_plots/simple_mpl.png'):
16         plt.clf()
17         plt.close()
18     plt.plot(*args: period, stat)
19     plt.plot(*args: period, norm_nums)
20     plt.savefig('simple_plots/simple_mpl.png')
21     t_mpl2 = datetime.datetime.now()
22     return f'{t_mpl2 - t_mpl1}'
23
```

Аналогичным образом создадим усложненный график, где исключим построение линии, отображающей норму, а также используем цикл for для построения линий всех параметров из таблицы.

```
23
24 # Строим усложненный график
25 def hard_mpl(period, stat_h): 2 usages  ⚡ Hazard-sakh +1 *
26     t_mpl1 = datetime.datetime.now()
27     if os.path.exists('complicated_plots/compl_mpl.png'):
28         plt.clf()
29         plt.close()
30     for i in stat_h:
31         plt.plot(*args: period, i)
32     plt.savefig('complicated_plots/compl_mpl.png')
33     t_mpl2 = datetime.datetime.now()
34     return f'{t_mpl2 - t_mpl1}'
35
```

## 3.2. Seaborn

Seaborn — это библиотека для создания статистической графики на Python. Она основана на Matplotlib и тесно интегрируется со структурами данных pandas.

Seaborn помогает изучать и понимать данные. Его функции построения графиков работают с `DataFrame`'ами и массивами, содержащими целые наборы данных, и выполняют необходимое семантическое сопоставление и статистическую агрегацию для создания информативных графиков. Его API, ориентирован на наборы данных, позволяет сосредоточиться на том, что означают различные элементы ваших графиков, а не на деталях их построения.

Seaborn для построения графиков неявно использует Matplotlib, создаёт полноценную графику с помощью одного вызова функции: при необходимости его функции автоматически добавляют информативные подписи к осям и легенды, которые объясняют семантические соответствия на графике.

Во многих случаях Seaborn также выбирает значения параметров по умолчанию в зависимости от характеристик данных.

Интеграция Seaborn с Matplotlib позволяет использовать его во многих средах, которые поддерживает Matplotlib, включая исследовательский анализ данных, взаимодействие в реальном времени в приложениях с графическим интерфейсом и архивирование в различных растровых и векторных форматах.

Сочетание высокоуровневого интерфейса Seaborn и широких возможностей настройки Matplotlib позволяет быстро изучать данные и создавать графику, которую можно адаптировать для публикации.

Seaborn был разработан специалистом по машинному обучению и нейробиологии компании Modal Labs Майклом Уаскомом (англ. Michael Waskom) и выпущен в 2014 году. Основная цель Seaborn — упростить создание сложных визуализаций и улучшить эстетику графиков, что делает его идеальным для быстрого анализа данных.

В академических кругах Seaborn часто применяется для визуализации результатов, которые затем публикуются в научных статьях и журналах. Графики, созданные с помощью неё, используются для демонстрации результатов экспериментов, а также для построения информативных графических абстракций, поддерживающих аргументацию исследований.

В бизнес-сфере Seaborn служит ключевым инструментом для визуализации данных, помогая аналитикам представлять сложные выводы в понятной форме для принятия решений. Графики, построенные с использованием этой библиотеки, могут украшать доклады и презентации, делая данные доступными для неподготовленной аудитории.

В сфере машинного обучения Seaborn используется для визуализации данных перед построением моделей, а также для анализа и интерпретации результатов моделирования. Она обладает функциями для построения матриц корреляции и многомерных распределений, что является ключевым этапом при предварительном анализе данных и в процессе создания новых признаков (характеристик) из имеющихся данных, чтобы улучшить работу модели машинного обучения.

Официальная документация размещена по адресу - <https://seaborn.pydata.org/>

Итак, приступим к созданию упрощенного графика с помощью библиотеки Seaborn. Как и при работе с Matplotlib начнем с импортирования библиотек `os` и `datetime` для работы с файлами и измерения быстродействия.

Затем импортируем Seaborn для построения графика и `matplotlib.pyplot` для манипуляций с ним.

Реализуем функцию, принимающую в себя ранее подготовленный `dataframe`, устанавливаем начало отсчета исполнения функции и условие, при котором график будет закрываться и удаляться если он уже имеется в указанной директории, обеспечив тем самым отсутствие перегрузки памяти и обновление графического представления графика.

Используя метод `.relplot` и передав в него `dataframe` и стиль в виде линий, создаем график, после чего методом `.savefig` сохраняем графическое представление графика в нужную директорию.

Завершаем функцию установкой окончания отсчета времени

```
by_seaborn.py x
1 import os
2 import seaborn as sns
3 import matplotlib.pyplot as plt
4 import datetime
5
6 # period - период для представления
7 # norm_name - название нормы
8 # norm_nums - список значений нормы по количеству периодов
9 # name - название объекта представления
10 # stat - статистика объекта представления
11
12
13 # Строим простейший график
14 def simple_sns(df): 2 usages 1 Hazard-sakh *
15     t_s1 = datetime.datetime.now()
16     if os.path.exists('simple_plots/simple_sns.png'):
17         plt.clf()
18         plt.close()
19     splt = sns.relplot(data=df, kind='line')
20     splt.savefig('simple_plots/simple_sns.png')
21     t_s2 = datetime.datetime.now()
22     return f'{t_s2 - t_s1}'
```

исполнения функции и возвращением с помощью оператора `return` значение функции в виде времени её работы.



Далее приступаем к созданию усложненного графика, в функцию реализации которого передаем ранее подготовленный dataframe, содержащий в себе сведения о всех параметрах для отображения за исключением значений нормирования.

Содержание функции абсолютно аналогично построению упрощенного графика: в коде меняется только переменная, отсылающая к dataframe.

```
23
24 # Строим усложненный график
25 def hard_sns(df_sns_h): 2 usages  ▲ Hazard-sakh +1 *
26     t_s1 = datetime.datetime.now()
27     if os.path.exists('complicated_plots/compl_sns.png'):
28         plt.clf()
29         plt.close()
30     splt = sns.relplot(data=df_sns_h, kind='line')
31     splt.savefig('complicated_plots/compl_sns.png')
32     t_s2 = datetime.datetime.now()
33     return f'{t_s2 - t_s1}'
34
```

### 3.3. Plotly

Plotly - это компания, занимающаяся техническими вычислениями, со штаб-квартирой в Монреале, которая разрабатывает онлайн-инструменты для анализа данных и визуализации. Plotly предоставляет онлайн-инструменты для построения графиков, аналитики и статистики для индивидуальных пользователей и совместной работы, а также научные графические библиотеки для Python, R, MATLAB, Perl, Julia, Arduino, JavaScript и REST.

Компания была основана Алексом Джонсоном, Джеком Пармером, Крисом Пармером и Мэтью Сандквистом.

Основатели имеют опыт работы в сфере науки, энергетики, анализа и визуализации данных. Среди первых сотрудников - Кристоф Вио, канадский инженер-программист, и Бен Постлтуэйт, канадский геофизик.

Plotly - одноименная библиотека с открытым исходным кодом, построенная на plotly.js, которая, в свою очередь, базируется на d3.js. Plotly поддерживает большое количество диаграмм, которые можно разделить на типы:

1. Базовые (точечный, линейный график, круговая диаграмма и т.д.)
2. Диаграммы искусственного интеллекта и машинного обучения (регрессия ML, классификация kNN, кривые ROC и PR)
3. Статистические диаграммы (гистограммы, дисплот, матрица диаграммы рассеяния, полосовая диаграмма, график предельного распределения)
4. Научные диаграммы (контурный график, тепловая карта, график параллельных координат, логарифмический участок, радиолокационная карта)
5. Финансовые графики (диаграмма водопада, график свечей, диаграмма воронки, калибровочная диаграмма)
6. Карты (линии на картах, пузырьковые карты, тепловые карты, заполненные области карты)
7. 3D-графики (точечный трехмерный график, трехмерный график поверхности, трехмерный сетчатый график, трехмерный линейный график, трехмерные кластерный и конусные графики)

При помощи Plotly легко сделать интерактивное представление графиков. Интерактивное представление позволяет не загромождать исходный график избыточной информацией, обращаясь к ней по необходимости.

Он также может создавать диаграммы, аналогичные Matplotlib и Seaborn, такие как линейные графики, точечные диаграммы, диаграммы с областями, столбчатые диаграммы и т. д.

Plotly также упрощает создание интерактивных графиков. Интерактивные графики не только красиво выглядят, но и позволяют публике более внимательно изучить каждую точку на графике. С помощью Plotly достаточно легко создавать сложные графики, он отлично подходит для создания интерактивных и качественных графиков при помощи нескольких строк кода.

Благодаря своей гибкости и простоте использования библиотека широко используется в различных областях: анализ данных, научные исследования, финансы, веб-разработка и других.

Официальная документация размещена на сайте - <https://plotly.com/python/>

Приступим к созданию упрощенного графика с использованием Plotly, для чего импортируем `os`, `datetime` и `plotly.graph_objs`.

Реализуем функцию построения графика по аналогии с функциями для предыдущих библиотек в части, касающейся определения быстродействия и обновления файла графического представления графика.

Как и в функции реализующей возможности Matplotlib включим аргументы, определяющие период, норму и статистику выбранного параметра, а также зададим линии графика, где в первой: X – период, Y – статистика выбранного параметра, и во второй: X – период, а Y – норма параметра.

Далее с применением метода `.write_image` сохраним графическое представление графика в нужную директорию.

Функция по исполнению возвращает значение времени.

```
by_plotly.py x
1 import os
2 import plotly.graph_objs as go
3 import datetime
4
5
6 # period - период для представления
7 # norm_name - название нормы
8 # norm_nums - список значений нормы по количеству периодов
9 # name - название объекта представления
10 # stat - статистика объекта представления
11
12
13 def simple_ptl(period, norm_nums, stat): 2 usages 1 Hazard-sakh +1 *
14     t_ptl1 = datetime.datetime.now()
15     fig = go.Figure()
16     fig.add_trace(go.Scatter(x=period, y=norm_nums, mode='lines'))
17     fig.add_trace(go.Scatter(x=period, y=stat, mode='lines'))
18     if not os.path.exists('simple_plots/'):
19         os.mkdir('simple_plots/')
20         if os.path.exists('simple_plots/simple_ptl.png'):
21             os.remove('simple_plots/simple_ptl.png')
22     name = 'simple_ptl'
23     fig.write_image(*args: f'simple_plots/{name}.png', format='png')
24     t_ptl2 = datetime.datetime.now()
25     return f'{t_ptl2 - t_ptl1}'
26
```

Аналогично строим уложенный график, за тем исключением, что в функцию его построения передаем аргумент, содержащий массив значений параметров для визуализации без значений о норме.

```
27
28 def hard_ptl(period, stat_h): 2 usages ± Hazar_sakh *
29     t_ptl1 = datetime.datetime.now()
30     fig = go.Figure()
31     for i in stat_h:
32         fig.add_trace(go.Scatter(x=period, y=i, mode='lines'))
33     if not os.path.exists('complicated_plots/'):
34         os.mkdir('complicated_plots/')
35         if os.path.exists('complicated_plots/complicated_plt.png'):
36             os.remove('complicated_plots/complicated_plt.png')
37     name = 'complicated_plt'
38     fig.write_image(f'complicated_plots/{name}.png')
39     t_ptl2 = datetime.datetime.now()
40     return f'{t_ptl2 - t_ptl1}'
41
```

Необходимо уделить особое внимание методу сохранения графического представления. Plotly для реализации этого метода неявно использует библиотеку kaleido, которая также должна быть установлена.

Вместе с тем у автора возникли трудности с работой данной библиотеки: графическое представление не сохранялось, среда разработки на некоторое время прекращала отзываться, исполнение программы останавливалось, принудительное прекращение заканчивалось ошибкой без описания. Вместе с тем при исполнении программы с применением ПК на рабочем месте автора таких трудностей не возникало.

В ходе исследования причин и возможных способов устранения возникшей проблемы были опробованы различные варианты написания кода, очистка кэша, переустановка среды разработки и интерпретатора. Указанные методы оказались безуспешны и привели к неоправданным затратам времени. Обращение к преподавателям тоже не принесло нужного результата.

Решение было найдено с помощью сообщения в англоязычном сегменте сайта Stack Overflow (<https://stackoverflow.com/questions/69016568/unable-to-export-plotly-images-to-png-with-kaleido>), где предлагалось использовать kaleido в версии 0.1.0.post1, чем автор работы и воспользовался.

В ходе последовавшей реализации работы программы возникла ошибка, содержащая информацию о том, что библиотека kaleido не может найти необходимые пакеты браузера Google Chrome.

После установки Chrome библиотека выдала ошибку о невозможности дальнейшей работы и необходимости установки более поздней версии, в связи с чем версия kaleido была обновлена до последней.

После описанных манипуляций, программа стала работать без сбоев и ошибок, графики, построенные с помощью Plotly, стали корректно сохраняться и отображаться в пользовательском интерфейсе.

## 4. Сравнительный анализ библиотек

В целях исполнения созданных функций построения графиков с применением пользовательского интерфейса, импортируем их в файл `main.py`.

```
1 from Backend.by_matplotlib import simple_mpl, hard_mpl
2 from Backend.by_seaborn import simple_sns, hard_sns
3 from Backend.by_plotly import simple_ptl, hard_ptl
```

Далее в функции, выполняемой по изменению (выбору) параметра из выпадающего списка, инициализируем выполнение функций построения графика.

Напомним, что функция построения графика возвращает время её исполнения, которую мы интегрируем в соответствующее текстовое поле пользовательского интерфейса.

Вместе с тем, в поля пользовательского интерфейса интегрируем масштабированную таблицу упрощенного графика.

Описанный алгоритм реализуем для каждой из упрощенных графиков.

```
74 # Выводим результаты построения упрощенного графика Matplotlib
75 self.ui.t_mpl_s.setText(simple_mpl(period, norm_nums, stat))
76 px_simple_mpl = QPixmap('simple_plots/simple_mpl.png')
77 self.ui.mpl_g_s.setScaledContents(True)
78 self.resize(px_simple_mpl.width(), px_simple_mpl.height())
79 self.ui.mpl_g_s.setPixmap(px_simple_mpl)
80
81 # Выводим результаты построения упрощенного графика Seaborn
82 self.ui.t_sns_s.setText(simple_sns(df))
83 px_simple_sns = QPixmap('simple_plots/simple_sns.png')
84 self.ui.sns_g_s.setScaledContents(True)
85 self.resize(px_simple_sns.width(), px_simple_sns.height())
86 self.ui.sns_g_s.setPixmap(px_simple_sns)
87
88 # Выводим результаты построения упрощенного графика Plotly
89 self.ui.t_ptl_s.setText(simple_ptl(period, norm_nums, stat))
90 px_simple_ptl = QPixmap('simple_plots/simple_ptl.png')
91 self.ui.ptl_g_s.setScaledContents(True)
92 self.resize(px_simple_ptl.width(), px_simple_ptl.height())
93 self.ui.ptl_g_s.setPixmap(px_simple_ptl)
```

После чего сделаем то же самое и для отображения усложненных графиков, и таким образом, завершим создание программы.

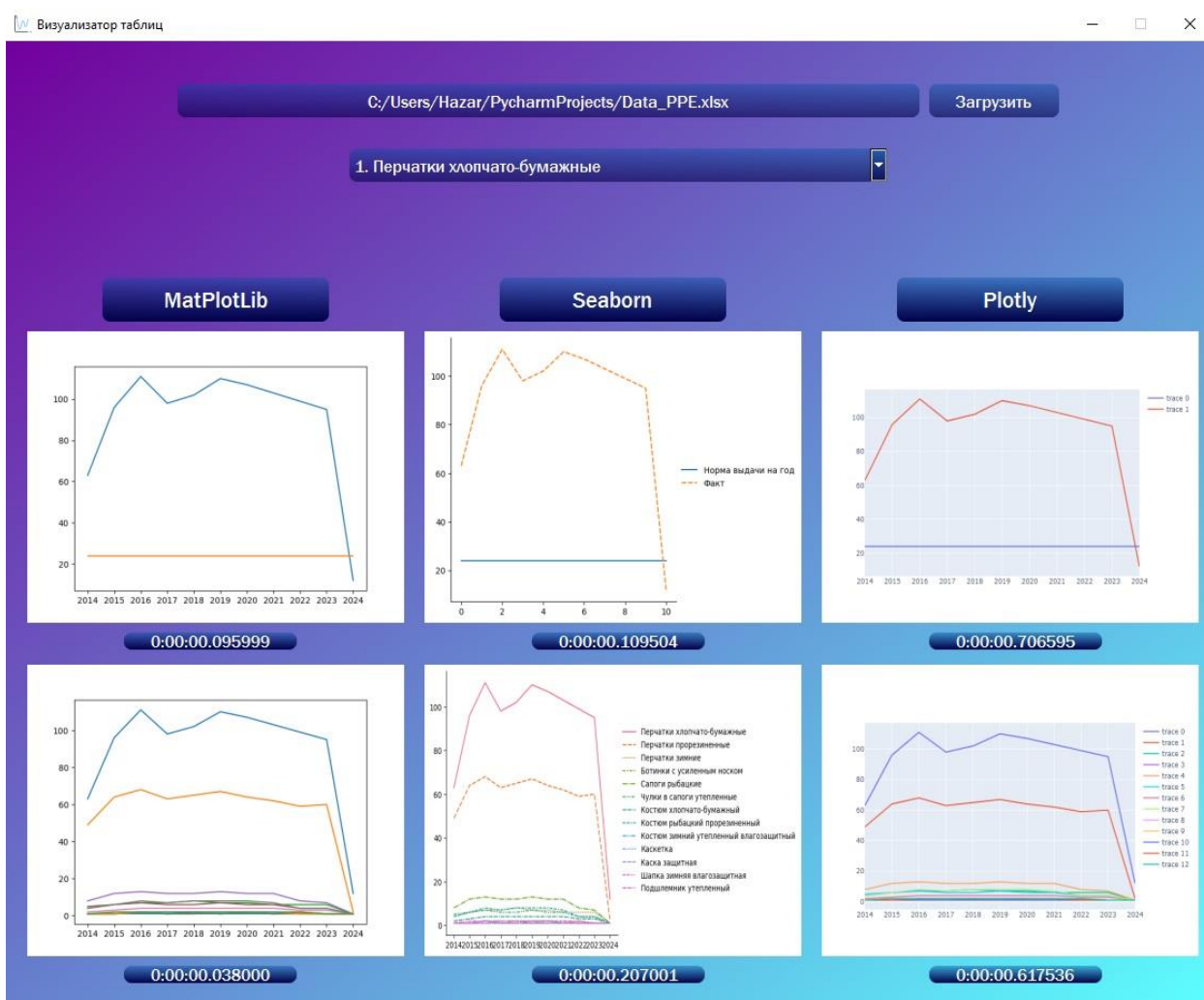


```

95 # Выводим результаты построения усложненного графика Matplotlib
96 self.ui.t_mpl_h.setText(hard_mpl(period, stat_h))
97 px_comp_mpl = QPixmap('complicated_plots/comp_mpl.png')
98 self.ui.mpl_g_h.setScaledContents(True)
99 self.resize(px_comp_mpl.width(), px_comp_mpl.height())
100 self.ui.mpl_g_h.setPixmap(px_comp_mpl)
101
102 # Выводим результаты построения усложненного графика Seaborn
103 self.ui.t_sns_h.setText(hard_sns(df_sns_h))
104 px_comp_sns = QPixmap('complicated_plots/comp_sns.png')
105 self.ui.sns_g_h.setScaledContents(True)
106 self.resize(px_comp_sns.width(), px_comp_sns.height())
107 self.ui.sns_g_h.setPixmap(px_comp_sns)
108
109 # Выводим результаты построения усложненного графика Plotly
110 self.ui.t_ptl_h.setText(hard_ptl(period, stat_h))
111 px_comp_ptl = QPixmap('complicated_plots/complicated_ptl.png')
112 self.ui.ptl_g_h.setScaledContents(True)
113 self.resize(px_comp_ptl.width(), px_comp_ptl.height())
114 self.ui.ptl_g_h.setPixmap(px_comp_ptl)

```

Для анализа работы библиотек проведем серию экспериментов с отображением различных параметров заданной таблицы.

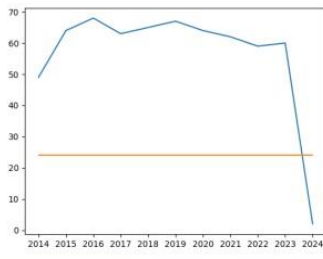


C:/Users/Hazar/PycharmProjects/Data\_PPE.xlsx

Загрузить

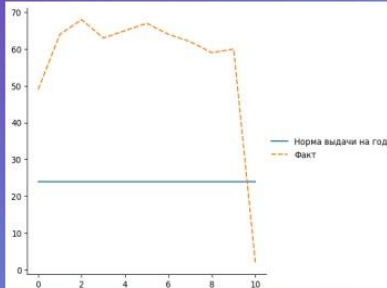
2. Перчатки прорезиненные

Matplotlib



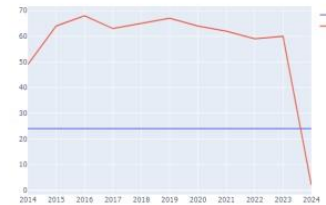
0:00:00.053000

Seaborn

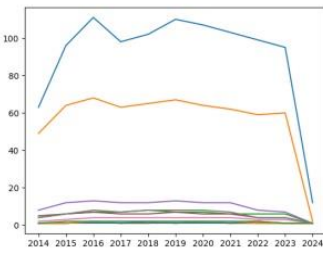


0:00:00.103000

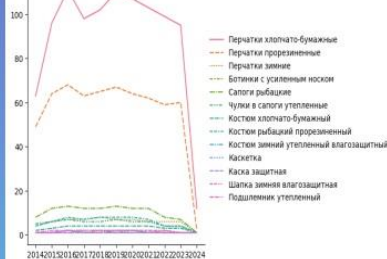
Plotly



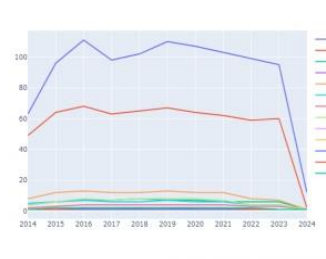
0:00:00.597607



0:00:00.036000



0:00:00.201000



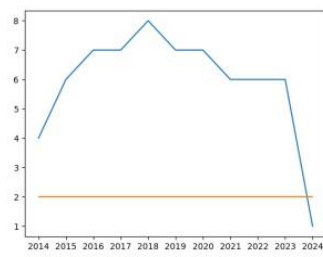
0:00:00.595909

C:/Users/Hazar/PycharmProjects/Data\_PPE.xlsx

Загрузить

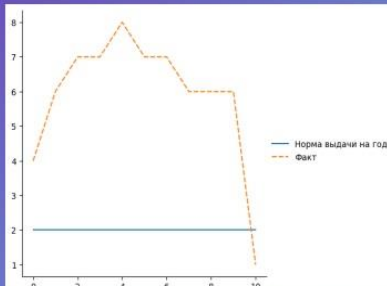
3. Перчатки зимние

Matplotlib



0:00:00.059999

Seaborn

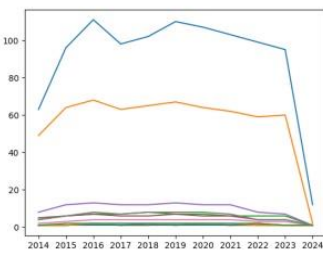


0:00:00.103000

Plotly



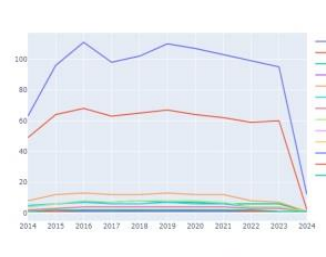
0:00:00.586309



0:00:00.042000



0:00:00.231003



0:00:00.602640



Итак, по результатам исполнения программы видим, что графики успешно создаются, при этом:

1. среднее время построения упрощенного графика с помощью Matplotlib по использованному методу составляет 0,066 секунды, и усложненного графика 0,038 секунды;
2. среднее время построения графика с помощью Seaborn составляет 0,105 секунды, усложненного – 0,211, следует отметить, что ввиду использования датафрейма в графике автоматически отображается легенда;
3. среднее время построения графика с помощью Plotly составляет 0,623 секунды, усложненного – 0,615, при этом, как и в Seaborn отображается легенда, но ввиду неприменения датафрейма отсутствует конкретизация.

Таким образом, оценивая библиотеки по определенным ранее критериям можно сделать вывод, что Seaborn в простейшей реализации передает всю смысловую нагрузку графика, дает возможность без дополнительной стилизации интерпретировать его и интегрировать в приложения. Это достигается за счет использования метода предобработки данных в датафрейм. Скорость его работы располагается в промежутке между результатами остальных сравниваемых библиотек, и является приемлемой. Вместе с тем, от пользователя требуются базовые знания библиотеки pandas

ввиду необходимости работы с датафреймами, в связи с чем данная особенность может иметь субъективную оценку пользователя как отрицательную (необходимость изучения pandas), так и положительную (построение графика из датафрейма с минимальными затратами времени на предобработку и получение приемлемого результата построения графика).

График, построенный с применением Plotly, менее информативен, что усложняет его восприятие без дополнительной стилизации, скорость его выполнения самая медленная, предобработка данных может быть реализована в различных вариантах. Особенности записи графического представления графика, связанные с необходимостью наличия библиотеки kaleido и браузера Google Chrome, являются скорее техническим недостатком, который компенсируется предназначением Plotly – построением интерактивных графиков высокого качества.

Matplotlib является наиболее быстродействующей библиотекой и в то же время без стилизации возможность интерпретация данных весьма сомнительна. Предобработка данных достаточно простая, не требует знаний сторонних библиотек, хотя возможность работы с датафреймами имеется.

Все три библиотеки, с т.з. простоты построения графиков без применения стилизации, оказались приемлемы. Официальная документация позволяет реализовать базовые методы в максимально краткие сроки. В то же время, особенности и возможности библиотек в большей степени определяют сферы их применения, нежели субъективные предпочтения пользователей.

## 5. Заключение

В рамках проведенного исследования были определены и оценены ключевые в рамках предложенной методики параметры библиотек визуализации Matplotlib, Seaborn и Plotly.

Вместе с тем, субъективные параметры, например, красота графиков, разнообразие тем и цветовых гамм, не учитывались, однако, для части пользователей они являются важными показателями для выбора используемой библиотеки.

В этом отношении Matplotlib является достаточно «гибкой» библиотекой, которую можно настроить очень подробно, и получить желаемый результат. Библиотека обладает большой коллекцией графиков. Построение красивого графика требует определенных затрат времени и среднего знания методов и атрибутов библиотеки, дизайнерских навыков для выбора приемлемых вариантов гамм и форм. Возможность взаимодействия с датафреймами имеется, но не является преимуществом, по сравнению с Seaborn.

Seaborn содержит в себе большое количество встроенных тем, имеет простой синтаксис для создания сложных графиков, автоматическое определение лучшего способа визуализации, интегрирован с pandas, что существенно облегчает работу с данными. Seaborn хоть и является наследником Matplotlib, и в некотором смысле, продвинутой оболочкой, упрощающей работу, однако для тонкой настройки требуется отличное знание библиотеки. Seaborn содержит меньшую коллекцию графиков по сравнению с Matplotlib, но графики, построенные с его помощью, при меньшем количестве кода выглядят визуально качественнее.

Plotly также упрощает создание интерактивных графиков. Интерактивные графики не только красиво выглядят, но и позволяют пользователю изучить график максимально подробно. С помощью Plotly достаточно легко создавать сложные графики – он имеет большую встроенную коллекцию графиков. Однако, настройка и создание графиков с его применением могут занимать много времени.

Стоит отметить, что Matplotlib, Seaborn и Plotly не являются взаимоисключающими и могут использоваться вместе в зависимости от конкретных потребностей проекта.

Matplotlib хорошо подходит для статичных графиков в печатных материалах, Seaborn прост в обращении и хорошо подходит для визуализации научных данных «на скорую руку», а Plotly отлично подходит для создания интерактивных и веб-визуализаций, создания карт.



В рамках бизнес-задачи по анализу расхода средств индивидуальной защиты в период с 2014 по 2024 годы наиболее предпочтительным решением будет использование Seaborn, однако если эту задачу необходимо было бы выполнить для интерактивного или веб-представления, то использование библиотеки Plotly видится более рациональным, так как Matplotlib хоть и обладает соответствующим функционалом, но реализация такого графика потребовала бы больших затрат времени и большего объема кода.

## 6. Приложение 1. Список необходимых библиотек

Проект реализован с помощью следующих библиотек:

pandas~=2.2.3

PySide6~=6.8.1

plotly~=5.24.1

seaborn~=0.13.2

matplotlib~=3.9.3

openpyxl~=3.1.5

kaleido~=0.2.1